Newlib

Gebruik van std C functies

- De GNU 68HC11 toolchain die wij gebruiken bevat ook een implementatie van **Newlib**
- **Newlib** is een std C library speciaal voor embedded systems. Bevat functies uit:
  - stdlib.h, ctype.h, stdio.h, string.h
  - signal.h, time.h, locale.h, stdarg.h
  - math.h, ...
- Maar... Functies zoals printf en malloc hebben OS-support nodig maar wij hebben geen OS
- Documentatie:
  - http://sources.redhat.com/newlib/libc.html
  - http://sources.redhat.com/newlib/libm.html
- Als voorbeeld zal ik siprintf gebruiken.

THRijswijk

siprintf

Printen naar een string

- sprintf
  - speciale versie van printf die niet naar stdout schrijft maar naar een "string" in het geheugen.
  - ANSI std C lib
- siprintf
  - speciale versie van sprintf die geen floating point getallen ondersteund.
  - geen ANSI std C lib (wel newlib)

```
#include "lcd.h"
#include <stdio.h>

void display(void) {
   char buffer[14];
   lcd_setpos(0,0);
   siprintf(buffer, "Snelheid = %2d", snelheid);
   lcd_puts(buffer);
// ...
```

THRijswijk

siprintf

Problemen!

- **make**:
  - stappenmotor.o: undefined reference to `sprintf'
- Waarom sprintf in plaats van siprintf?
- De std C lib (newlib) wordt niet automatisch meegelinkt (bij "gewoon" gcc wel!)
  - makefile aanpassen:

```
...
LIBS      = -L. -lsimlcd -lc
...
```

THRijswijk

siprintf

Problemen!

- **make**:
  - region rom is full (a.out section .text)
  - region ram is full (a.out section .data)
- memory map van de simulator aanpassen File, Options, Memory Configuration...
  - RAM: $0000-$0FFF
  - ROM: $4000-$FFFF
- evm.ld vervangen door sim.ld
  - sim.ld aanpassen:

```
...
MEMORY
{
  ram    (rwx): ORIGIN=0x0000, LENGTH=0x1000
  rom     (rx): ORIGIN=0x4000, LENGTH=0xc000
}
/* Setup the stack on the top of the internal RAM. */
PROVIDE (_stack = 0x0fff);
```

  - makefile aanpassen:

```
...
LDSCRIPT   = sim.ld
```

THRijswijk

siprintf

Problemen!

- **make**:
  - libc.a: undefined reference to `isatty'
  - idem voor `sbrk', `write', `close', `fstat', `lseek', `read'
- Documentatie siprintf:
  - Supporting OS subroutines required: close, fstat, isatty, lseek, read, sbrk, write. Waarom?
- 68HC11 GNU toolchain bevat ook de library libnosys.a (niet gedocumenteerd!)
  - makefile aanpassen:

```
...
LIBS      = -L. -lsimlcd -lc -lnosys
...
```

THRijswijk

siprintf

Problemen!

- **make**:
  - libnosys.a(sbrk.o): undefined reference to `end'
  - HACK! sim.ld aanpassen:

```
...
PROVIDE (_end = .);
PROVIDE (end = .); /* HACK! om -lnosys te
                      kunnen gebruiken */
```

- **Memory size** (footprint)
  - RAM 1330 bytes (was 23) **53x**
  - ROM 18499 bytes (was 1480) **12x**

THRijswijk

-mshort

- De 68HC11 gcc compiler heeft de optie -mshort
  - *Consider type int to be 16 bits wide, like short int*
  - Deze optie heeft ook tot gevolg dat alle berekeningen in 16 bits worden uitgevoerd! ANSI C zet namelijk de operanden van een rekenkundige expressie altijd eerst om naar int. Dit heet: "integer promotion".
  - timer->toc1+=10000/snelheid;
    - timer->toc1 => unsigned short
    - snelheid => unsigned char
    - zonder -mshort => 32 bits deling
    - met -mshort => 16 bits deling
  - Dit heeft consequenties voor het stappenmotor programma:
    - zonder -mshort => ISR is niet snel genoeg bij snelheid = 14 of 15 Oplossing?
    - met -mshort => ISR is altijd snel genoeg!

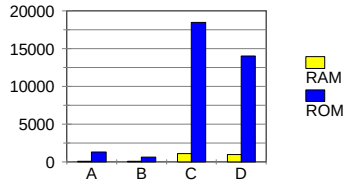
THRijswijk

-mshort

- De optie -mshort beinvloed ook de footprint: !readelf -l a.out
  - oorspronkelijke stappenmotor.c:
    - zonder -mshort: RAM: 23 ROM: 1330
    - met -mshort: RAM: 17 ROM: 620
  - met siprintf:
    - zonder -mshort: RAM:1132 ROM:18499
    - met -mshort: RAM: 980 ROM:14053
- De libraries lcd en simlcd moeten ook met -mshort gecompileerd worden. Anders:
  - Error: linking files compiled for 16-bit integers (-mshort) and others for 32-bit integers

THRijswijk

-mshort



A. orginele stappenmotor.c
B. met -mshort
C. siprintf
D. siprintf met -mshort

THRijswijk

Vraag

```
typedef unsigned char byte;
volatile byte* portb=(byte*)0x1004;
volatile byte* tmsk2=(byte*)0x1024;
volatile byte* tflg2=(byte*)0x1025;

volatile byte x=0;
volatile byte y=0;

void rtti_isr(void) __attribute__((interrupt));
void rtti_isr(void) {
   ++x;
   ++y;
   *tflg2=0x40;
}

int main() {
   *tmsk2|=0x40;
   *portb=0x00;
   while (1) {
      if (x!=y)
         ++*portb;
   }
   return 0;
}
```

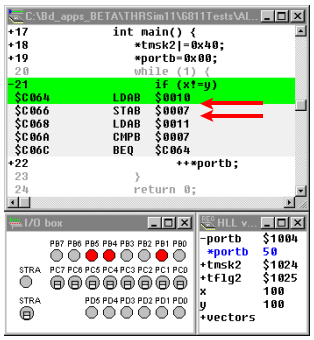
Gaan de LED's branden?

THRijswijk

80

Probleem

-Os -fomit-frame-pointer



THRijswijk

81

Oplossingen

```
int main() {
   *tmsk2|=0x40;
   *portb=0x00;
   while (1) {
      *tmsk2&=~0x40;
      if (x!=y)
         ++*portb;
      *tmsk2|=0x40;
   }
   return 0;
}
```

Zet **rtti** bij vergelijken even uit!

Zet **alle interrupts** bij vergelijken even uit!

```
int main() {
   *tmsk2|=0x40;
   *portb=0x00;
   while (1) {
      asm ("sei");
      if (x!=y)
         ++*portb;
      asm ("cli");
   }
   return 0;
}
```

THRijswijk

82

C => C

```
short add(short a, short b, short* c) {
   *c=a+b;
   if ((a>0 && b>0 && *c<0) ||
       (a<0 && b<0 && *c>0))
      return 1;
   return 0;
}
```

add_c.c

```
int main() {
   short add(short a, short b, short* c);
   // test add
   volatile short i=30000;
   volatile short j=10000;
   volatile short overflow=0;
   short resultaat;
   overflow=add(i, j, &resultaat);
   while (1);
   return 0;
}
```

test_overflow.c

```
OBJECTS = test_overflow.o add_c.o
```

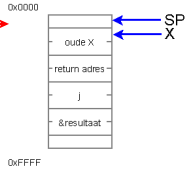
makefile

THRijswijk

83

C => ASM

```
        .global add
        .text
add:    pshx
        tsx
        addd  4,x
        bvs   vset
        ldx   6,x
        std   0,x
        ldd   #0
        bra   return
vset:   ldd   #1
return: pulx
        rts
        .end
```



add_s.s

idem!
Bij aanroep overflow=add(i, j, &resultaat); wordt i in register D doorgegeven. De rest via de stack. Returnwaarde via register D.

test_overflow.c

```
OBJECTS = test_overflow.o add_s.o
```

makefile

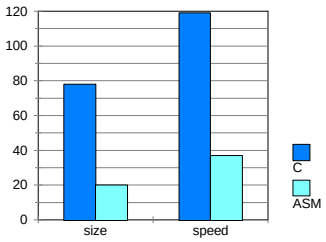
THRijswijk

84

C versus ASM

Functie add

- ! C code gecompileerd met: -O3 -fomit-frame-pointer -mshort
- ! Size in bytes, Speed in clock-cycles



THRijswijk

85

C => ASM

```
int main() {
   short i=0;
   extern short fun(short, short);
   i=fun(i, 0x00bd);
   i=fun(i, 0xbc43);
   return 0;
}
```

prog.c

```
      .text
      .global fun
fun:  pshx
      tsx
      addd  4,x
      pulx
      rts
```

fun.s

```
OBJECTS = prog.o fun.o
...
%.o : %.c
   $(GCC) -g -c -m68hc11 -Wall $(GCC_OPTIONS) $<

%.o : %.s
   $(AS) --gdwarf2 -m68hc11 $(AS_OPTIONS) -o $*.o $<
```

makefile

THRijswijk

86

C => ASM

Waarom moet de offset voor het ophalen van de tweede parameter 4 zijn?

- ! X is bij binnenkomst op de stack gezet (2 bytes)
- ! PC is bij aanroepen van de functie op de stack gezet (2 bytes)

C++ => ASM

```
extern "C" short fun(short, short);

int main() {
   short i(0);
   i=fun(i, 0x00bd);
   i=fun(i, 0xbc43);
   return 0;
}
```

prog.cpp

THRijswijk

87